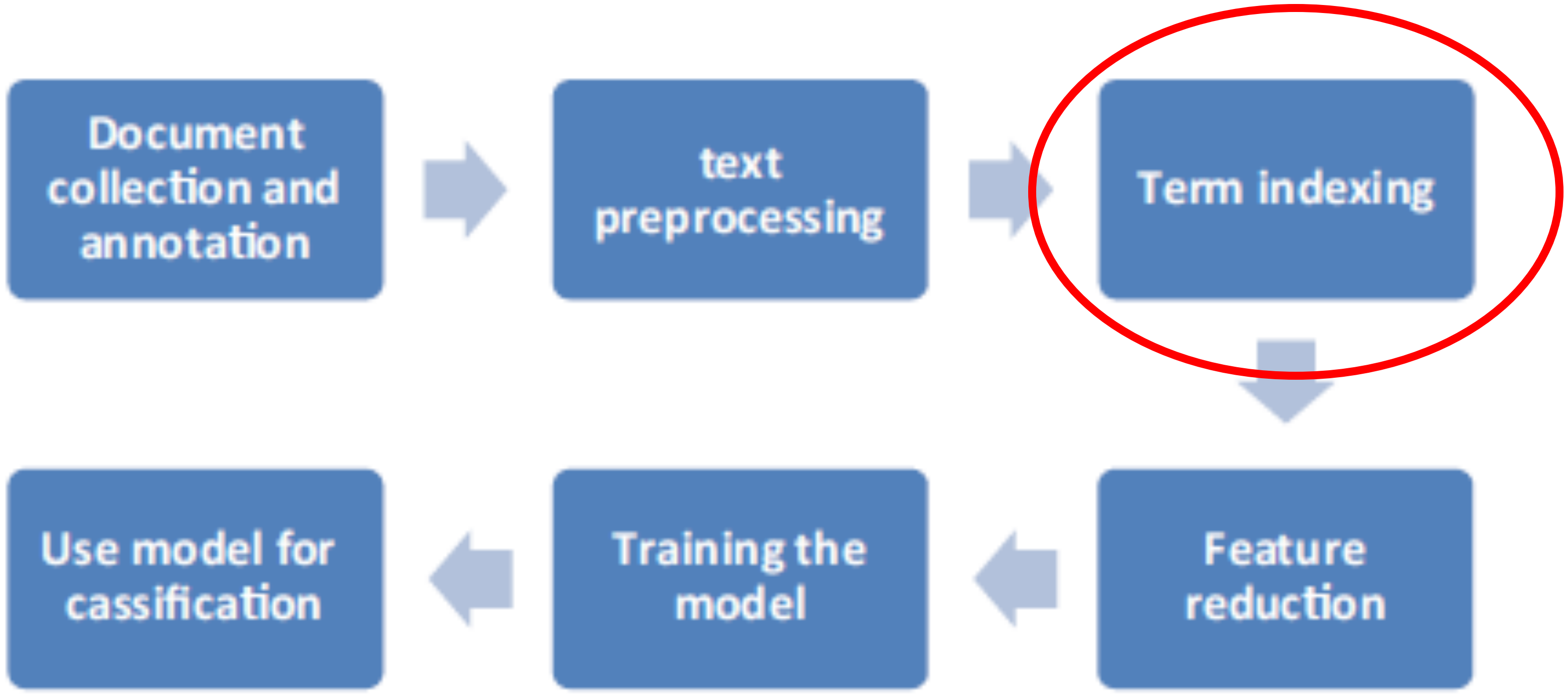
Machine Learning Implementations in Arabic Text Classification

Marwa F. Mohamed

Text Classification Steps



Indexing

- The process of tagging or associating information with a file so it can be used for **search and retrieval** purposes later.
- Indexing creates the “searchable” information that users will later use to find documents.

Document Indexing

- Is the process of associating information with a file or specific tag for **search and retrieval purposes** in the future.
- The Indexed information will then be programmed into a database / document management system, which helps users to easily access the data they require.
- **Without** implementing a document indexing process, the ability to retrieve documents can be very time consuming and costly.
- From this perspective, **vector space representation** can be considered a document indexing method.

Text Vectorization

- Machine learning with natural language is faced with one major hurdle – its algorithms usually deal with numbers, and natural language is, well, text.
- So we need **to transform that text into numbers**, otherwise known as **text vectorization**.
- It's a fundamental step in the process of machine learning for analyzing data, and different vectorization algorithms will drastically affect end results, so you need to choose one that will deliver the results you're hoping for.

Text Vectorization

- **TF-IDF** enables us to gives us a way to associate each word in a document with a number that **represents how relevant each word is in that document.**
- Then, documents with similar, relevant words will have similar vectors, which is what we are looking for in a **machine learning algorithm.**

Text Vectorization

- **Text Vectorization** refers to a set of **feature extraction techniques in Natural Language Processing (NLP)**.
- Most importantly **aims** at transforming words into numbers and text documents into high dimensional vector space model.
- It is especially helpful in some of the most important applications of **text classifiers like finding spam/not spam messages.**

TF-IDF

- **TF-IDF** stands for Term Frequency Inverse Document Frequency of records.
- It can be defined as the calculation of how relevant a word in a series or corpus is to a text.
- **TF-IDF** is one of the popular and most widely used **term weighting technique in today’s IR system.**

Terminology

- t — term (word)
- d — document (set of words)
- corpus — the total document set
- N — count of corpus

Term Frequency

- This measures the frequency of a word in a document.

  - **tf(t,d) = count of t in d / number of words in d**

Document Frequency

- This measures the importance of document in whole set of **corpus**.
- This is very similar to TF.
- The only **difference is that**
  - **TF** is frequency counter for a term t in document d,
  - where as **DF** is the count of occurrences of term t in the document set N.

Document Frequency

- ***In other words,*** **DF** is the number of documents in which the word is present.
- We consider one occurrence if the term consists in the document at least once, we do not need to know the number of times the term is present.
  - **df(t) = occurrence of t in documents**

Inverse Document Frequency

- **IDF** is the inverse of the document frequency which measures the *informativeness of term t*.
- When we calculate IDF, it will be **very low** for the most occurring words such as **stop words.**
- Because **stop words** such as "is" is present in almost all of the documents, and **N/df** will give a very **low value** to that word.
- This finally gives what we want, a **relative weightage**.
  - **idf(t) = N/df**

Inverse Document Frequency

- In case of a large corpus, say 10,000, the IDF value **explodes**.
- So to dampen the effect we take the logarithm of IDF.
  - **idf(t) = log(N/df)**

Inverse Document Frequency

- **idf(t) = log [n / df(t)] + 1**
- The effect of adding "1" to the idf in the equation above is that terms with zero idf, i.e., terms that occur in all documents in a training set, will not be entirely ignored.

Inverse Document Frequency

- The constant “1” is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents **zero divisions:**
  - **idf(t) = log [(1 + n) / (1 + df(t))] + 1.**

Example: Inverse Document Frequency

- Let us identify the individual values for the words — ***potential*** and ***wings***.
  - Number of Documents = 8
  - Number of documents in the corpus that contain the word ***potential*** = 1
  - Number of documents in the corpus that contain the word ***wings*** = 2
- Applying the formula for Inverse Document Frequency, we get

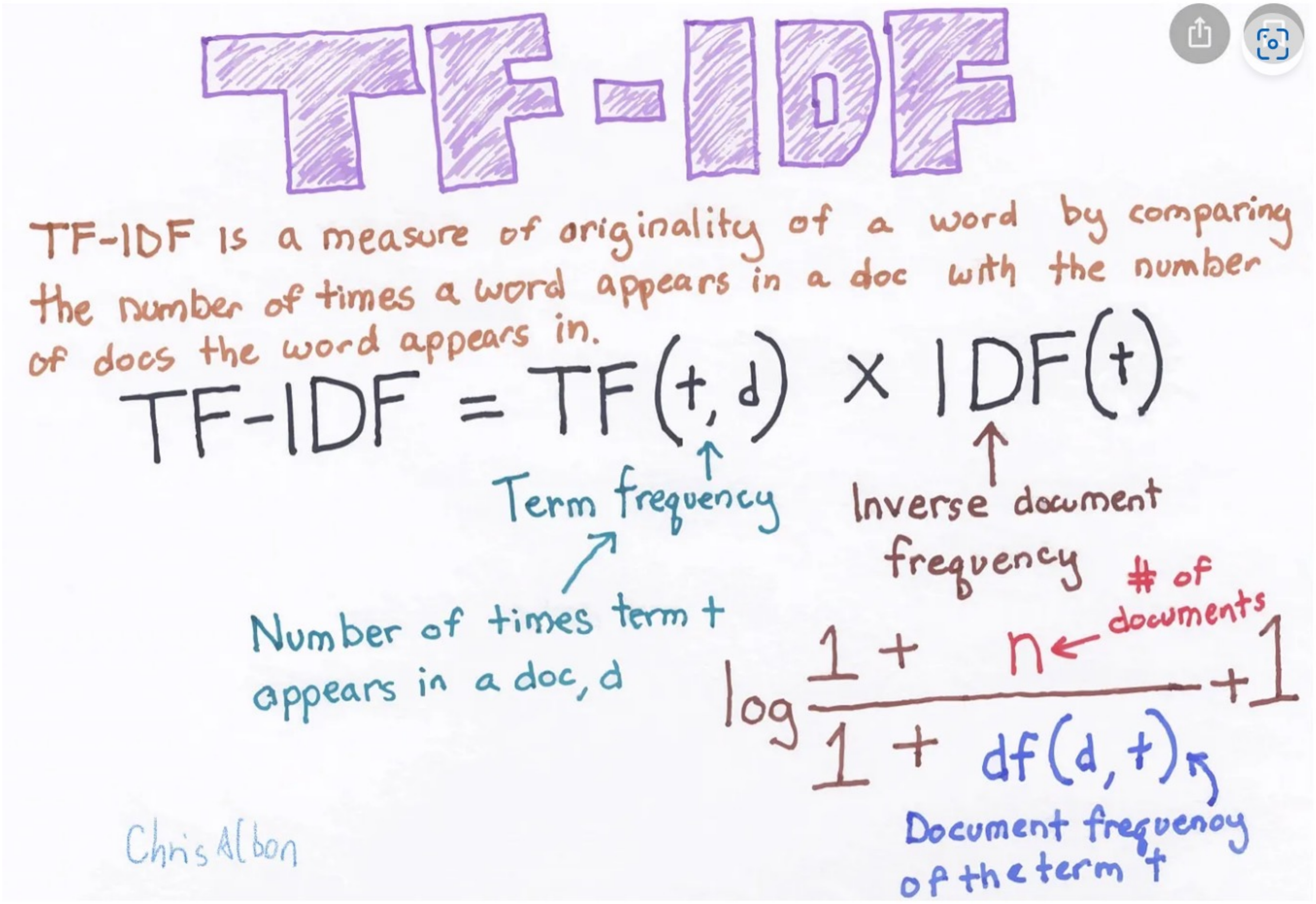
Inverse Document Frequency

$$idf(potential) = \ln(\frac{1+8}{1+1}) + 1 = \ln(\frac{9}{2}) + 1 = 2.504077$$

$$idf(wings) = \ln(\frac{1+8}{2+1}) + 1 = \ln(\frac{9}{3}) + 1 = 2.098612$$

TF-IDF

- The **tf-idf** measure allows us to evaluate the importance of a term to a document.
- The importance is **proportional** to the number of times the term appears in the document **(TF)** but is offset by the frequency of the documents in which the term has occurred **(IDF).**



TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

$$\log \frac{1 + n}{1 + df(d, t)} + 1$$

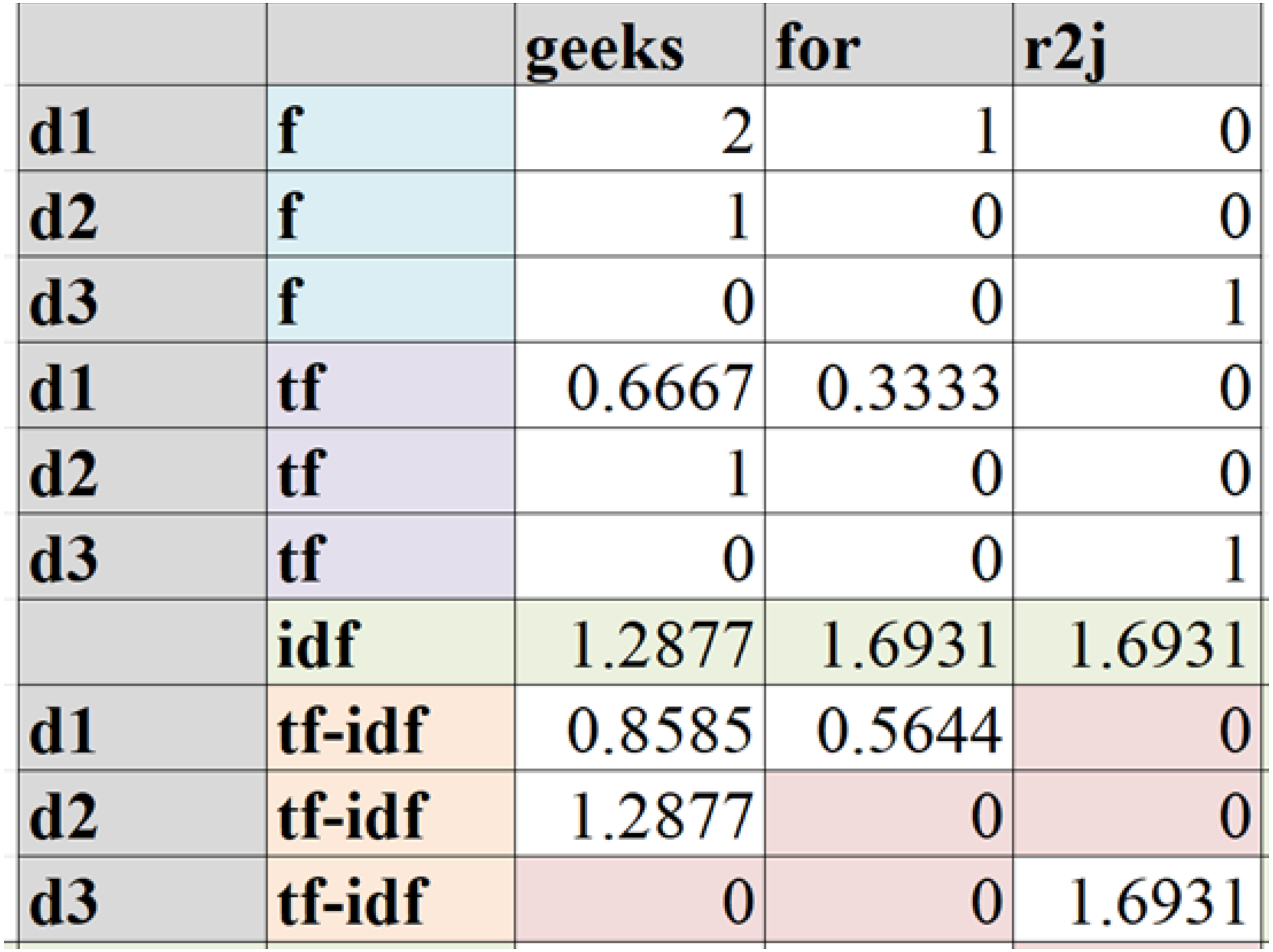
Term frequency
Number of times term t appears in a doc, d
Inverse document frequency
# of documents
Document frequency of the term t
Chris Albon

TF-IDF

- **Tf-idf** is one of the best metrics to determine how significant a term is to a text in a series or a corpus.
- **Tf-idf** is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (tf) (idf).
- The **words with higher scores** of weight are deemed to be more significant.
- In python **tf-idf** values can be computed using ***TfidfVectorizer()*** method in ***sklearn* module**.

Example 1

- # assign documents
- d1 = 'Geeks for geeks’
- d2 = 'Geeks’
- d3 = 'r2j'



		geeks	for	r2j
d1	f	2	1	0
d2	f	1	0	0
d3	f	0	0	1
d1	tf	0.6667	0.3333	0
d2	tf	1	0	0
d3	tf	0	0	1
	idf	1.2877	1.6931	1.6931
d1	tf-idf	0.8585	0.5644	0
d2	tf-idf	1.2877	0	0
d3	tf-idf	0	0	1.6931

Normalize the product of TF and IDF

- To apply L2 norm, for each of the sentences we need to **calculate the square root of the sum of squares** of the product of TF and IDF.

$$||v|| = \sqrt{v_1{}^2 + v_2{}^2 + v_3{}^2 \ldots \ldots \ldots v_n{}^2}$$

Normalize the product of TF and IDF

- **Sum of square for d1** =SQRT($0.8585^2+0.5644^2$) =

  1.027361662

- **TF-IDF (geeks)** = 0.8585 / 1.027361662 = 0.8355915

- **TF-IDF (for) =** 0.5644 / 1.027361662 = 0.5493512

			geeks	for	r2j	
	d1	f	2	1	0	
	d2	f	1	0	0	
	d3	f	0	0	1	
	d1	tf	0.6667	0.3333	0	
	d2	tf	1	0	0	
	d3	tf	0	0	1	
		idf	1.2877	1.6931	1.6931	SumofSquare
	d1	tf-idf	0.8585	0.5644	0	1.02736166
	d2	tf-idf	1.2877	0	0	1.28768207
	d3	tf-idf	0	0	1.6931	1.69314718
	d1	tf-idf	0.835592	0.549351	0	
	d2	tf-idf	1	0	0	
Normalized	d3	tf-idf	0	0	1	

```python
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign documents
d0 = 'Geeks for geeks'
d1 = 'Geeks'
d2 = 'r2j'

# merge documents into a single corpus
string = [d0, d1, d2]

# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)

# get idf values
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)

# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf value:')
print(result)

# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```

```
idf values:
for : 1.6931471805599454
geeks : 1.2876820724517808
r2j : 1.6931471805599454

Word indexes:
{'geeks': 1, 'for': 0, 'r2j': 2}

tf-idf value:
  (0, 0)        0.5493512310263033
  (0, 1)        0.8355915419449176
  (1, 1)        1.0
  (2, 2)        1.0

tf-idf values in matrix form:
[[0.54935123 0.83559154 0.        ]
 [0.         1.         0.        ]
 [0.         0.         1.        ]]
```

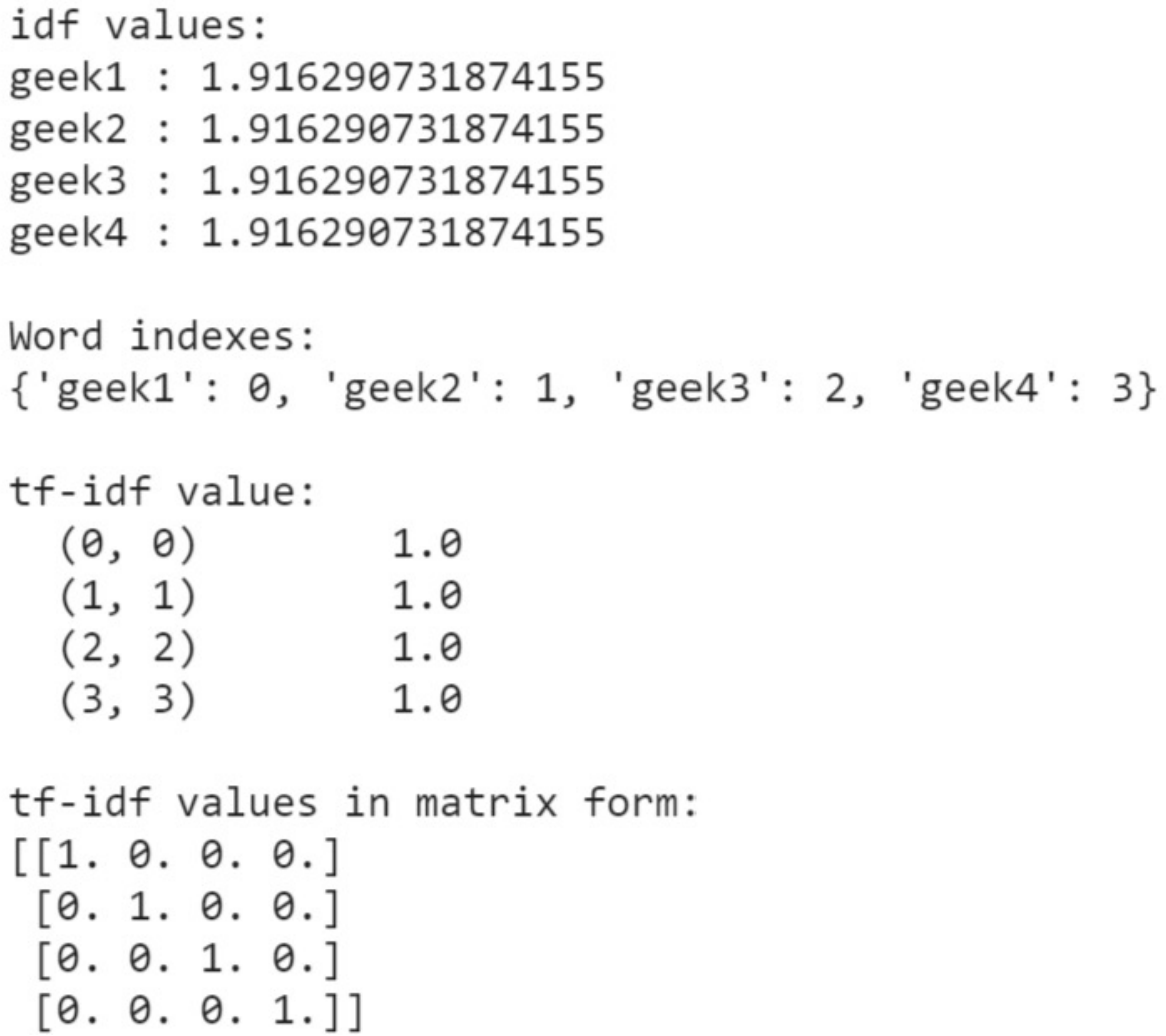
Example 2

- **tf-idf** values are computed from a **corpus having unique values.**
- # assign documents
- d0 = 'geek1'
- d1 = 'geek2'
- d2 = 'geek3'
- d3 = 'geek4'

		geek1	geek2	geek3	geek4
d0	f	1	0	0	0
d1	f	0	1	0	0
d2	f	0	0	1	0
d3	f	0	0	0	1
d0	tf	1	0	0	0
d1	tf	0	1	0	0
d2	tf	0	0	1	0
d3	tf	0	0	0	1
	idf	1.9163	1.9163	1.9163	1.9163
d0	tf-idf	1.9163			
d1	tf-idf		1.9163		
d2	tf-idf			1.9163	
d3	tf-idf				1.9163
d0	Ntf-idf	1	0	0	0
d1	Ntf-idf	0	1	0	0
d2	Ntf-idf	0	0	1	0
d3	Ntf-idf	0	0	0	1

```python
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer
# assign documents
d0 = 'geek1'
d1 = 'geek2'
d2 = 'geek3'
d3 = 'geek4'
# merge documents into a single corpus
string = [d0, d1, d2, d3]
# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)
# get idf values
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)
# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)
# display tf-idf values
print('\ntf-idf value:')
print(result)
# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```



```
idf values:
geek1 : 1.916290731874155
geek2 : 1.916290731874155
geek3 : 1.916290731874155
geek4 : 1.916290731874155

Word indexes:
{'geek1': 0, 'geek2': 1, 'geek3': 2, 'geek4': 3}

tf-idf value:
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0

tf-idf values in matrix form:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Example 3

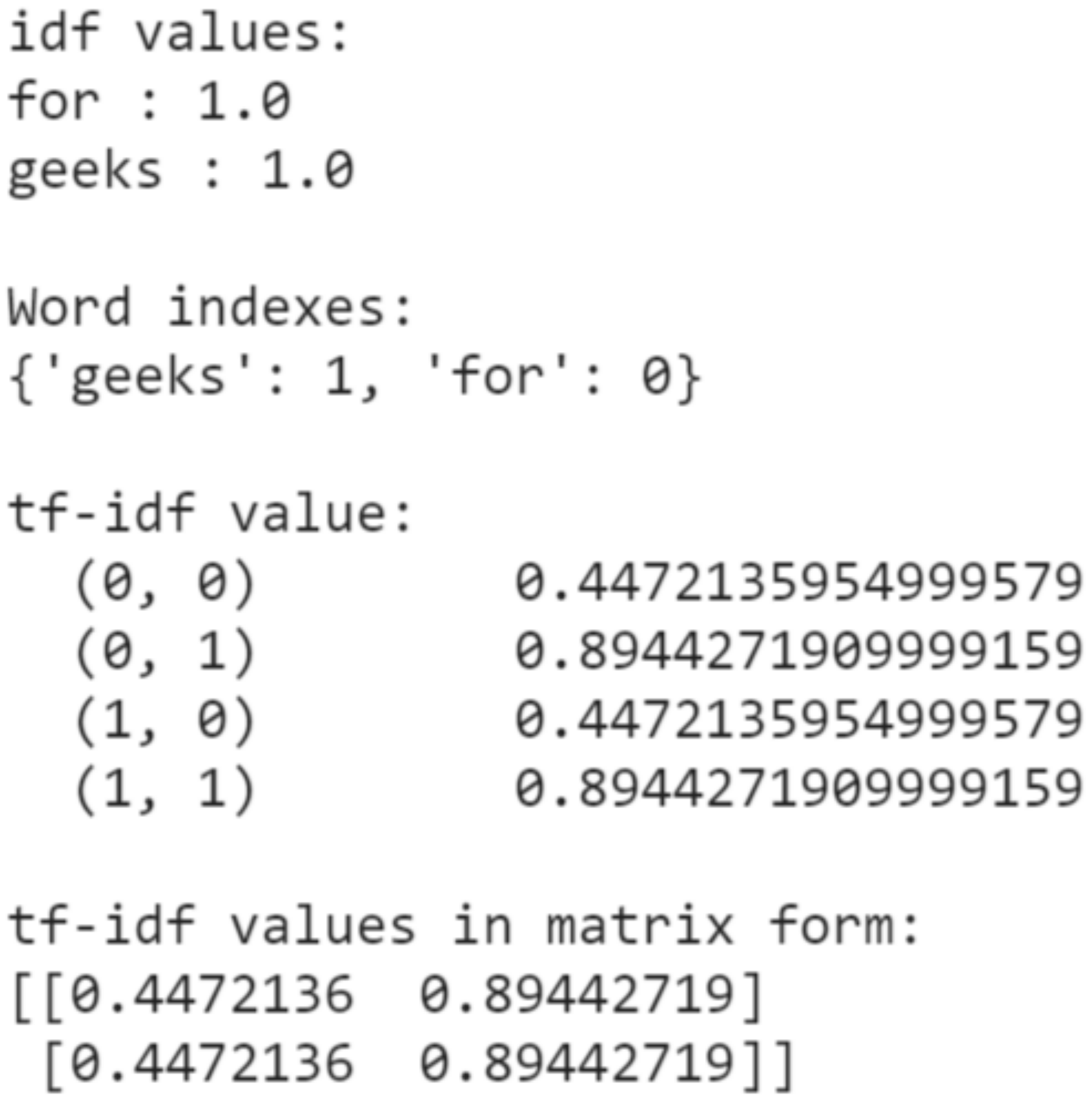
- **tf-idf values** are computed from a corpus having similar documents.
- # assign documents
- d0 = 'Geeks for geeks!'
- d1 = 'Geeks for geeks!'

		geeks	for	
d0	f	2.00	1.00	
d1	f	2.00	1.00	
d0	tf	0.67	0.33	
d1	tf	0.67	0.33	
	idf	1.00	1.00	
d0	tf-idf	0.6667	0.3333	0.745356
d1	tf-idf	0.6667	0.3333	0.745356
d0	Ntf-idf	0.8944	0.4472	
d1	Ntf-idf	0.8944	0.4472	

```python
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer
# assign documents
d0 = 'Geeks for geeks!'
d1 = 'Geeks for geeks!'

# merge documents into a single corpus
string = [d0, d1]
# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)
# get idf values
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)
# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)
# display tf-idf values
print('\ntf-idf value:')
print(result)
# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```



```
idf values:
for : 1.0
geeks : 1.0

Word indexes:
{'geeks': 1, 'for': 0}

tf-idf value:
  (0, 0)        0.4472135954999579
  (0, 1)        0.8944271909999159
  (1, 0)        0.4472135954999579
  (1, 1)        0.8944271909999159

tf-idf values in matrix form:
[[0.4472136  0.89442719]
 [0.4472136  0.89442719]]
```

Applications of TF-IDF

- **Information retrieval**

- TF-IDF was invented for document search and can be used to deliver results that are most relevant to what you're searching for.
- Imagine you have a search engine and somebody looks for LeBron.
- The results will be displayed in order of relevance. That's to say the most relevant sports articles will be ranked higher because TF-IDF gives the word LeBron a higher score.
- It's likely that every search engine you have ever encountered uses TF-IDF scores in its algorithm.

References

- Understanding TF-IDF (Term Frequency-Inverse Document Frequency) – GeeksforGeeks
- Understanding TF-ID: A Simple Introduction (monkeylearn.com)
- sklearn.feature_extraction.text.TfidfTransformer — scikit-learn 1.2.2 documentation
- A Gentle Introduction To Calculating The TF-IDF Values | by Ann Sebastian | Towards Data Science